Q1

0 Points

You can find the entire Midterm 1 Policies posted on ed here. You are expected to have already read those policies, but we have copied the most pertinent information over from the post.

Resources / Cheating Policy

This is an open-book, open internet, open IDE, and **closed-friend test**. You may consult any books, notes, or online resources available to you, **but all code must be written by you alone**. You may use any program / text supplied in lectures, labs, or tutoring sections. **However, you may not consult other students, friends, acquaintances, or online strangers for answers.** Such prohibited acts include, but are not limited to, posting on forums like StackOverflow, messaging other students, creating a group chat with other students, posting public comments on ed, or pushing to any shared repository. We will also be shutting down all CS 61BL related discords for the duration of the exam period. If you do any of these actions (or other unlisted but similar actions) during the exam window, we will treat them as academic dishonesty.

Please do not discuss this exam with people until we announce it is okay to do so. Discussing with others will be treated as academic dishonesty. As stated in our course policies for exams, **we will be absolutely unforgiving if you cheat on an exam.** Any incident will result in a **failing grade for the course**, though Berkeley will let you retake CS 61B next semester. **All incidents of exam cheating will be referred to the Office of Student Conduct.** Every semester we penalize a significant number of people. Do not be one of them. Heed the advice of those who came before you.

We will be running our cheating detection software against your submissions. We vastly prefer providing you an exam that is open for 24 hours and does not include live proctoring, as we feel it will lend to a testing environment that more accurately assesses your programming abilities and is in general more equitable. However, we **will** change these policies if we are concerned about the validity of student scores after assessing the results. This will lead to future exams being more stressful both for you and the course staff.

Obviously, the expressive power of Java is a subset of the English language. And yes, you can obviously obey the letter of this entire policy while completely violating its spirit. However, **this policy is not a game to be defeated**, and such circumventions will be seen as plagiarism.

If you do commit any act which we might deem to be academic dishonesty, immediately notify us by emailing cs61bl@berkeley.edu. If you come forward before we catch you, the consequences might be made less severe. If we catch you and you have not confessed, again we will strictly enforce the above policy.

Exam Format and Submission

You will submit your assignment through two separate mechanisms, gradescope and git. You should be familiar with submitting assignments through both of these methods before. The process will be mostly the same as what you have seen on labs, quizzes, and projects.

Gradescope

As specified above you will start the exam through gradescope. Once you start the assignment, you should see something similar to what you might have seen on quizzes (although the length and number of questions will differ). All instructions for how to complete each question will be provided through this gradescope assignment.

Some questions will be submitted entirely through gradescope. For these you should select your answers / type them directly into gradescope. Other questions will be submitted through git, but the instructions and any necessary starter code will be provided in the gradescope assignment.

Git

The questions which will need to be submitted through git will be clearly marked on the exam. For each of these questions you will need to modify corresponding `*.java` files in the `mt1` directory of your **personal repo (your repo with the name su20-s***).**

There will be an autograder running on gradescope which will allow you to verify compilation and API for the required files, but it will run no correctness tests (this will resemble the Project 1 autograder, but we will not run style for the midterm i.e. style will not be a part of the grade).

You should submit to this assignment again by adding, committing, pushing, tagging and pushing your tags, as you have all summer. For the midterm you should submit with the tag `mt1-n` where `n` is some integer. You may submit to this autograder as many times as you like, but we will only be grading your most recent submission within the allotted time.

The autograder will be open for the entire 24 hour exam window, but we will retroactively be enforcing lateness. We will flag your submissions if you submit to the assignment before you begin your exam or if you submit after your exam window has closed. You **must** submit your code before the end of your testing window on gradescope (e.g. before time runs out on gradescope) otherwise **you will not receive credit for the coding questions.**

If for whatever reason, you are unable to submit your code through git, you may email us your code by emailing the files to cs61bl@berkeley.edu (again you must send this email before time runs out on gradescope). However, submitting through git is part of your grade, and **failure to submit through git will result in a 10% deduction on all coding questions.** Git **is** in scope and you are expected to be comfortable using it at this point.

Skeleton Code

For every coding question which you are required to submit through git, we will provide a corresponding empty, but correctly named, file through the `skeleton-su20` remote. Each question will specify clearly which file needs to be edited. All such files will be included in the `mt1` directory. To receive these skeleton files, you will need to pull from the skeleton remote (again you should be familiar with how to do this by now).

You must type your solutions into these specified files, or the autograder will not recognize your submission. Additionally, all of your solutions that you write must go into these files. You may create additional files which may contain tests, but they will not be included in your submission when we run the autograder. Finally, any discussion of the skeleton code before we specify that you can talk about the exam will be considered academic dishonesty and will be penalized in accordance with the policy outlined above.

Points

Your exam should contain 5 problems numbered Q2 through Q6. Officially, it is worth 18 points (out of a total of 300). Point values have been indicated in the

title of each question.

Enter your su20-s* number (e.g. if you are su20-s123 you should enter in 123) to confirm you have read the policies above.**

Course ID Number (enter your 1-3 digit number):

I pledge my honor that during this examination, I have neither given nor received assistance. All code written has been written by me alone. I understand that failure to follow these polices (and those listed in the ed post) will result in penalty as outlined by the policies.

Full Name:

By entering your full name you are signing that you agree to the above statement.

Q2 BuggyDeque

3 Points

Provided below is a buggy implementation of some methods from Project 1: Deques (there may be a bug in one or both of the methods). Assume that all other methods not shown have been implemented and function as expected. For your convenience we have also included the interface `Deque.java`

```
public interface Deque<T> {
    void addFirst(T x);
    void addLast(T x);

    default boolean isEmpty() {
        // implementation not shown
    }

    int size();
    void printDeque();
    T removeFirst();
    T removeLast();
```

```
    T get(int index);
}
```

```
1  public class ArrayDeque<T> implements Deque<T> {
2
3      private T[] items;
4      private int size;
5      private int nextFirst;
6      private int nextLast;
7      private int START_SIZE = 8;
8
9      public ArrayDeque() {
10         // implementation not shown
11         // assume all instance variables are correctly instantiated
12     }
13
14     public T removeFirst() {
15         if (size == 0) {
16             return null;
17         }
18         if (items.length > START_SIZE && size <= 0.25 * items.length) {
19             resize(items.length / 2);
20         }
21         size -= 1;
22         int index = wrapIndex(nextFirst + 1);
23         items[index] = null;
24         T item = items[index];
25         nextFirst = index;
26         return item;
27     }
28
29     private int wrapIndex(int index) {
30         return (index + items.length) % items.length;
31     }
32
33     // other method implementations not shown
34 }
```

Q2.1 Bug Identification

1 Point

Identify and describe the bug(s). Be sure to include the line number(s) which contains the error(s), a clear description of why the line(s) have issues, and additionally how the bug(s) could be fixed.

E.g. your answer should be something like "The bug(s) is <description of the bug(s)> in line(s) <line(s)> and we could fix it by doing ...".

Q2.2 Testing

2 Points

Write a JUnit test for `ArrayDeque` which should pass given a correct implementation, but will fail if ran on the above buggy implementation.

Assume that you are writing the test in `ArrayDequeTest.java` and that all necessary JUnit libraries have been imported.

```
import org.junit.Test;
import static org.junit.Assert.*;

public class ArrayDequeTest {
```

```
}
```

Q3 Pointer and Box

3 Points

For the two parts of this question, you will be asked to complete the main methods. After the execution of the main method (but before any pointers are lost), the state of the program will match that of the box and pointer diagram depicted below in each part. Assume that all lines provided are properly executed before the lines you provide.

You will be limited by the number of lines that you can provide. Each box should contain at most one statement, e.g. at most one semicolon per box. Additionally multiple assignments are prohibited, e.g. at most one `=` per box. Failure to adhere to this will result in a loss of points. Not all boxes may be needed.

The two subquestions will also refer to the following two classes, but each subquestion (3.1 and 3.2) will be independent from one another.

```
public class Point {
    public int x;
    public int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

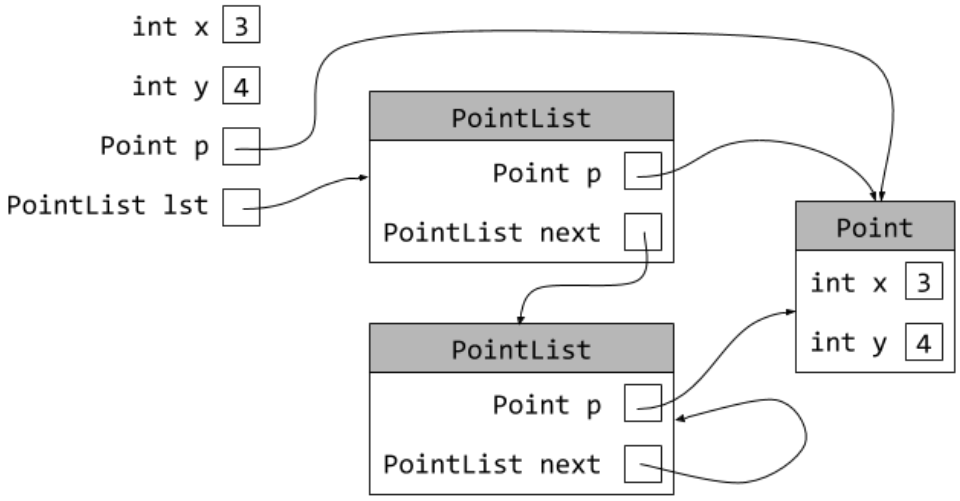
```
public class PointList {
    public Point p;
    public PointList next;

    public PointList(Point p, PointList next) {
        this.p = p;
        this.next = next;
    }
}
```

Q3.1 Pointer and Box 1

1.5 Points

Below is the box and pointer diagram after execution of the `main` method. Fill in the missing lines. The main method (with your solution included) **must compile and run without error.**

Remember there should only be **one statement (e.g. one semicolon) per box**. Additionally multiple assignments are prohibited, **e.g. at most one `=` per box.**




```
public static void main(String[] args) {
    int x = 3;
    int y = 4;
    Point p = new Point(x, y);
```

```
    PointList lst = new PointList(p, new PointList(p, null));
```

```
    lst.next.next = lst.next;
```

```
}
```

Q3.2 Pointer and Box 2

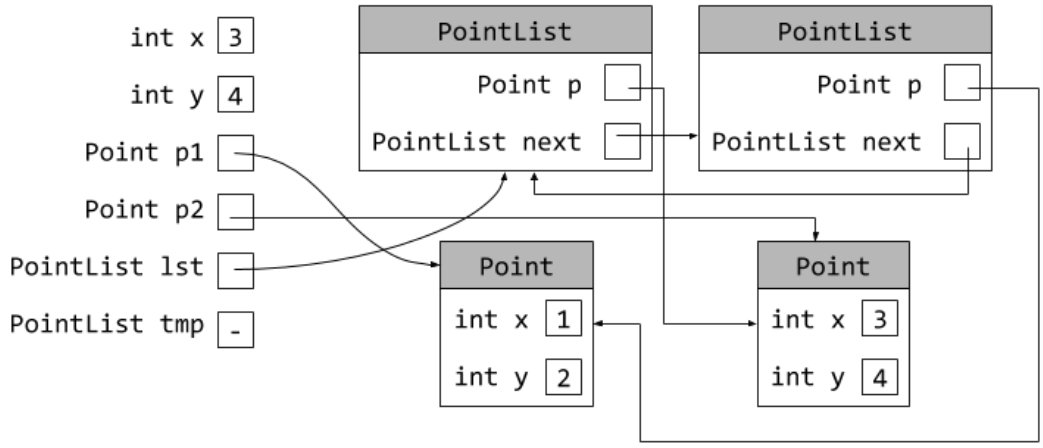
1.5 Points

Below is the box and pointer diagram after execution of the `main` method. Fill in the missing lines. The main method (with your solution included) **must compile and run without error.**

Remember there should only be **one statement (e.g. one semicolon) per box**. Additionally multiple assignments are prohibited, **e.g. at most one `=` per box.**

There is one more restriction on 3.2: in order to get full credit you may not use the `new` keyword more than once across both boxes. If you use the `new` keyword more than once across both boxes you will receive 0 points for this sub question.

Also note that there are lines both before and after the lines you will provide. All lines will be executed before the state of the program is shown in the image.




```
public static void main(String[] args) {
    Point p1 = new Point(1,2);
    Point p2 = new Point(3,4);
    PointList lst = new PointList(p2, null);
```

temp = new PointList (p1, lst)

lst.next = temp

```
    int x = tmp.next.next.next.p.x;
    int y = lst.p.y;
    tmp = null;
}
```

Q4 Step to the Beat

4 Points

GIT SUBMISSION: This question will be submitted through git. Full instructions are listed above. The file required for this question will be `IntArrayListStepper.java`.

This question will deal with the following implementation of an `IntArrayList`.

```
public class IntArrayList {

    public IntArrayList() {
        // implementation not shown
    }

    public void setList(int... nums) {
        // implementation not shown
    }

    public int get(int i) {
        // implementation not shown
    }

    public int size() {
        // implementation not shown
    }

    public void add(int e) {
        // implementation not shown
    }
}
```

One of your friends has implemented the class `IntArrayListStepper` which serves as a wrapper around a given `IntArrayList`. This means an instantiation of an `IntArrayListStepper` class would include an instantiated `IntArrayList` instance variable. `IntArrayListStepper` also has the instance variable `int step` where `step >=0`. A partial implementation of the `IntArrayListStepper` function is shown below.

```
public class IntArrayListStepper {

    private IntArrayList lst;
    private int step;

    public IntArrayListStepper(IntArrayList lst, int step) {
        this.lst = lst;
```

```
        this.step = step;
    }

    /** Returns the ith element in the IntArrayListStepper. This should
     *  use the lst.get() method and should step through this underlying
     *  IntArrayList by the instance variable step. The value of i passed
     *  in can also be negative where the lst will be stepped through in
     *  reverse starting from the last element. Assume that the value i
     *  will be in bounds for lst. Refer to the JUnit tests
     *  provided to specify the behavior further.
     */
    public int get(int i) {
        // TODO: Implement this method.
    }

}
```

Your task for this question will be to fill in the `get` method for `IntArrayListStepper`. A description of this has been provided above the method in the `IntArrayListStepper` class. For this question, you may assume that the argument `i` passed in to `get` will not overstep the bounds of the stored `lst`. In other words, you do not need to perform any bounds checking or otherwise handle these cases.

Finally, we will be checking the performance of the code. Your solution for `get` should run in worst case $\Theta(1)$ time. Solutions that run in any runtime slower than worst case $\Theta(1)$ will be penalized.

We also **strongly** recommend that you read over the following JUnit tests which specify the behavior of this method.

```
import org.junit.Test;
import static org.junit.Assert.*;

public class IntArrayListStepperTest {

    @Test
    public void stepByZeroTest() {
        IntArrayList lst = new IntArrayList();
        lst.setList(1,2,3,4);

        IntArrayListStepper stepper = new IntArrayListStepper(lst, 0);
        assertEquals(stepper.get(0), 1);
        assertEquals(stepper.get(1), 1);
        assertEquals(stepper.get(2), 1);
        assertEquals(stepper.get(3), 1);
        assertEquals(stepper.get(50), 1);
```

```
    }

    @Test
    public void stepByOneTest() {
        IntArrayList lst = new IntArrayList();
        lst.setList(1,2,3,4);

        IntArrayListStepper stepper = new IntArrayListStepper(lst, 1);
        assertEquals(stepper.get(0), 1);
        assertEquals(stepper.get(1), 2);
        assertEquals(stepper.get(2), 3);
        assertEquals(stepper.get(3), 4);
    }

    @Test
    public void stepByOneNegativeTest() {
        IntArrayList lst = new IntArrayList();
        lst.setList(1,2,3,4);

        IntArrayListStepper stepper = new IntArrayListStepper(lst, 1);
        assertEquals(stepper.get(-4), 1);
        assertEquals(stepper.get(-3), 2);
        assertEquals(stepper.get(-2), 3);
        assertEquals(stepper.get(-1), 4);
    }

    @Test
    public void stepByThreeTest() {
        IntArrayList lst = new IntArrayList();
        lst.setList(1,2,3,4,5,6,7,8,9);

        IntArrayListStepper stepper = new IntArrayListStepper(lst, 3);
        assertEquals(stepper.get(0), 1);
        assertEquals(stepper.get(1), 4);
        assertEquals(stepper.get(2), 7);
    }

    @Test
    public void stepByThreeNegativeTest() {
        IntArrayList lst = new IntArrayList();
        lst.setList(1,2,3,4,5,6,7,8,9);

        IntArrayListStepper stepper = new IntArrayListStepper(lst, 3);
        assertEquals(stepper.get(-3), 3);
        assertEquals(stepper.get(-2), 6);
        assertEquals(stepper.get(-1), 9);
    }
}
```

You should start with the provided starter code for `IntArrayListStepper` above. If you have not already, pull from the skeleton remote. You should copy this code into the file `mt1/IntArrayListStepper.java` in your personal

`su20-s***` course repository and then submit through git. See more submission instructions at the top of the exam.

Finally we will disallow a number of classes (of which the full list will not be provided). You should not need to import any class in your solutions. When you submit to the gradescope assignment, **make sure that you do not have any errors in the "Illegal Dependency Check" section** (note if you do not have any errors you should not see this section.

Q5 CS 61BasePairs

6 Points

GIT SUBMISSION: This question will be submitted through git. Full instructions are listed above. The file required for this question will be `DNASequenceSet.java`***.***

For this problem, you will be required to create a data structure to help with a DNA sequencing project. In order to do so, we will need to create a data structure called `DNASequenceSet` which will be similar to the linked lists that we have been using with a few changes.

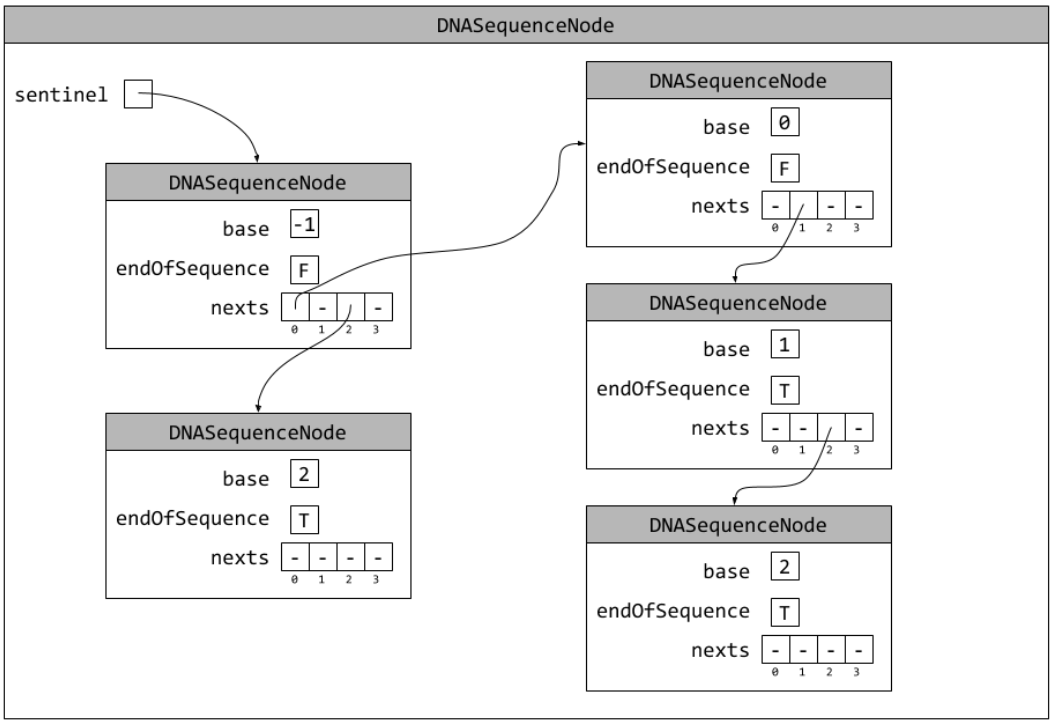
For this problem we will assume that the four base pairs of DNA can be represented by the numbers `0`, `1`, `2`, and `3`. A DNA sequence will therefore be represented by an `int[]` where all of the integers contained will be either `0`, `1`, `2`, or `3`.

There will be two methods in the `DNASequenceSet` class, `public void add(int[] sequence)` and `public boolean contains(int[] sequence)`. The `add` method will allow you to add an additional sequence to the `DNASequenceSet` and the `contains` method will return `true` if the given sequence has previously been added (else will return false).

The suggested implementation for this will be as follows. Previously our linked lists had just one next pointer. In order to more compactly store the base pairs, we can notice that many of them might overlap. Given this we will store the contained base pairs in an overlapping fashion. Each `DNASequenceNode` will have four next pointers which correspond to each of the four possible base pairs (`0`, `1`, `2`, or `3`). These can be compactly stored as an `DNASequencePair[]`

with four elements (the index will correspond to each of the next possible base pairs). When either inserting a sequence or checking if it will be contained you will thus have multiple paths possible through this new data structure. As we will have multiple overlapping paths, we also will need to mark which of these `DNASequenceNodes` correspond to valid sequences which have been added (we will do so with the boolean `endOfSequence`).

Below is a visual representation of this structure. This particular `DNASequenceSet` contains **only** the sequences `[0,1,2]`, `[0,1]` and `[2]`.



Finally, we will also be checking the performance of the code. Your solution for both `add` and `contains` should run in worst case $\Theta(\ell)$ where ℓ is the length of `sequence` passed in. Solutions that run in $\Theta(N)$ where N is the total number of sequences added or some slower runtime than worst case $\Theta(\ell)$, will be penalized. Note that if you follow the implementation we have suggested, you should not have to worry about this.

You should start with the provided starter code for `DNASequenceSet` (including the inner class `DNASequenceNode`) above. If you have not already, pull from the skeleton remote. You should copy this code into the file `mt1/DNASequenceSet.java` in your personal `su20-s***` course repository and

then submit through git. See more submission instructions at the top of the exam.

Again we will disallow a number of classes (of which the full list will not be provided). You should not need to import any class in your solutions. When you submit to the gradescope assignment, **make sure that you do not have any errors in the "Illegal Dependency Check" section** (note if you do not have any errors you should not see this section.

Note that you may choose to use any amount of our provided code. You should not change the API or class name, but the implementations can be rewritten from scratch if you so desire.

```
public class DNASequenceSet {
    private DNANode sentinel;

    public DNASequenceSet() {
        sentinel = new DNANode(-1, false);
    }

    public void add(int[] sequence) {
        DNANode pointer = sentinel;
        for (int i = 0; i < sequence.length; i++) {
            ____________________;
            if (____________________ == null) {
                ____________________ = new DNANode(____________________, false);
            }
            pointer = ____________________;
        }
        pointer.endOfSequence = true;
    }

    public boolean contains(int[] sequence) {
        DNANode pointer = sentinel;
        for (int i = 0; i < sequence.length; i++) {
            ____________________;
            if (____________________ == null) {
                return ____________________;
            }
            pointer = ____________________;
        }
        return ____________________;
    }


    public static class DNANode {
        private int base;
        private boolean endOfSequence;
        private DNANode[] nexts;
```

```
        public DNANode(int b, boolean end) {
            this.base = b;
            this.endOfSequence = end;
            this.nexts = new DNANode[4];
        }
    }
}
```

In order to test your implementation feel free to use the following unit tests. These are a direct subset of the unit tests that we will be running on your code, e.g. passing these tests mean you will get at least some fraction of the points.

```
import org.junit.Test;
import static org.junit.Assert.*;

public class DNASequenceSetTest {
    @Test
    public void testSingleBase() {
        DNASequenceSet set = new DNASequenceSet();
        set.add(new int[]{0});
        assertTrue(set.contains(new int[]{0}));
        assertFalse(set.contains(new int[]{1}));
    }

    @Test
    public void testAllSameTwo() {
        DNASequenceSet set = new DNASequenceSet();
        set.add(new int[]{0,0});
        assertTrue(set.contains(new int[]{0,0}));
        assertFalse(set.contains(new int[]{0}));
    }

    @Test
    public void testMultipleOverlapping1() {
        DNASequenceSet set = new DNASequenceSet();
        set.add(new int[]{0,1});
        assertFalse(set.contains(new int[]{0}));
        assertTrue(set.contains(new int[]{0,1}));

        set.add(new int[]{3,2,1,0});
        assertFalse(set.contains(new int[]{0}));
        assertTrue(set.contains(new int[]{0,1}));
        assertFalse(set.contains(new int[]{3}));
        assertFalse(set.contains(new int[]{3,2}));
        assertFalse(set.contains(new int[]{3,2,1}));
        assertTrue(set.contains(new int[]{3,2,1,0}));
    }
}
```

Q6 Loose Ends and Tidbits

2 Points

Q6.1 Switcheroo

1 Point

In your own words, why does the following test fail?

```
import org.junit.Test;
import static org.junit.Assert.*;

public class Error {

    public static void swap(int x, int y) {
        int temp = x;
        x = y;
        y = temp;
    }

    @Test
    public void testSwap() {
        int a = 1;
        int b = 2;
        swap(a, b);
        assertEquals(a, 2);
        assertEquals(b, 1);
    }
}
```

Q6.2 61BL Cast a Spell on Me

1 Point

Suppose we have the following three classes.

```
public class A {
    // implementation not shown
}
```

```
public class B extends A {
    // implementation not shown
}
```

```
public class C extends A {
    // implementation not shown
}
```

We also have the following main method.

```
public static void main(String[] args) {
    A aa = new A();
    B bb = new B();
    A ac = new C();

    C c1 = (C) aa; // line 1
    C c2 = (C) bb; // line 2
    C c3 = ac;     // line 3
}
```

Suppose we only included line 1 (e.g the lines marked `line 2` and `line 3` were commented out). In this case our code will compile but there will be a runtime exception. Explain both why the code compiles and why there is an exception.

Now suppose we only included line 2 (e.g the lines marked `line 1` and `line 3` were commented out). In this case our code will not compile. Explain why this code will not compile.

Now suppose we only included line 3 (e.g the lines marked `line 1` and `line 2` were commented out). In this case our code will not compile. Explain why this code will not compile.

We can alter line 3 such that it compiles, runs and properly assigns the object pointed to by `ac` to the `c3` variable (again assume the lines marked `line 1` and `line 2` are commented out). What would be the corrected version of this line (your answer should be the corrected line of code)?

Midterm 1

● UNGRADED

357 DAYS, 18 HOURS LATE

STUDENT
Unknown Student (removed from roster?)

TOTAL POINTS
- / 18 pts

QUESTION 1
(no title) 0 pts

QUESTION 2
BuggyDeque 3 pts

2.1 Bug Identification 1 pt

2.2 Testing 2 pts

QUESTION 3
Pointer and Box 3 pts

3.1 Pointer and Box 1 1.5 pts

3.2 Pointer and Box 2 1.5 pts

QUESTION 4